Reconocimiento y Clasificación de Objetos en Paralelo

A.C. Rodrigo Felice[1], A.C. Fernando Ruscitti[1], Lic. Marcelo Naiouf[2], Ing. Armando De Giusti[3]

Laboratorio de Investigación y Desarrollo en Informática[4]
Departamento de Informática - Facultad de Ciencias Exactas
Universidad Nacional de La Plata

Resumen

Se presenta el desarrollo de algoritmos paralelos para el reconocimiento y clasificación automática de objetos que provienen de una línea industrial (de producción o packaging). Este tipo de problema introduce una restricción temporal sobre el procesamiento de las imágenes, la cual naturalmente obliga a pensar en la resolución paralela del mismo.

Se eligieron objetos simples (frutas, huevos), los cuales son clasificados tomando en cuenta características tales como forma, color, tamaño, defectos (manchas, decoloraciones), etc. Con esta clasificación los objetos pueden ser derivados, por ejemplo, a distintas vías de la línea de producción o empaquetado.

Se estudia la paralelización de los algoritmos sobre una red heterogénea de computadoras y soporte PVM (Parallel Virtual Machine).

Finalmente, se presentan algunos resultados cuantitativos obtenidos al aplicar el algoritmo sobre una muestra representativa de imágenes.

[1] Analista de Computación. Alumno Avanzado de la Licenciatura en Informática. Dpto. de Informática, Fac. de Ciencias Exactas, UNLP.

[2] Prof. Adjunto Dedidcación Exclusiva. LIDI. Dpto. de Informática, Fac. de Ciencias Exactas, UNLP. E-mail: mnaiouf@info.unlp.edu.ar

[3] Inv. Principal CONICET. ProfesorTitular. Dedicación Exclusiva, Dpto. de Informática, Fac. Ciencias Exactas, UNLP. E-mail: degiusti@info.unlp.edu.ar

[4] Calle 50 y 115 - 1er piso - (1900) La Plata - Buenos Aires
Tel/Fax: +54 21 227707
E-mail: lidi@info.unlp.edu.ar

1. Introducción

Existe un creciente interés en la automatización de procesos de manipulación y toma de decisiones delegados hasta ahora casi exclusivamente a la mano y el discernimiento del hombre. Una gama muy amplia de los mismos se basa en la percepción visual del mundo que nos rodea, por lo que su automatización requiere la adquisición y procesamiento de imágenes, y tiene relación con el área de la informática que se ocupa de Visión por Computadora [Har92][Jai95].

Cuando los procesos de clasificación no se encuentran automatizados son llevados a cabo mediante distintos métodos:

- Mecánicos. Sólo son utilizables si los objetos son geométricamente regulares. Además, pueden ser imprecisos, y sólo permiten diferenciar por tamaños.
- Electrónicos. Están orientados a la detección de colores, y son muy específicos para una aplicación. Se utilizan fotocélulas que envían información a controladores lógicos programables, los cuales la procesan.
- Manuales. Personal entrenado realiza la clasificación visualmente, por lo que sufren las limitaciones humanas: son inexactos, dependen del entrenamiento, están sujetos al criterio personal, y además es difícil la realización de tareas rutinarias durante largos periodos, y la eficiencia se degrada con el tiempo, siendo necesario reemplazar el personal a intervalos relativamente cortos, incrementando los costos de la clasificación.

Los sistemas de visión por computadora tratan de imitar el proceso de visión humano para poder automatizar tareas que tienen un alto grado de dificultad en cuanto a exactitud y tiempo, o son excesivamente rutinarias. Su utilización es muy amplia ya que los avances tecnológicos provocan la automatización de procesos hasta ahora desarrollados por alguno de los métodos nombrados.

El objetivo de un sistema de visión por computadora es crear un modelo del mundo real a partir de imágenes: recuperar información útil acerca de una escena a partir de sus proyecciones 2D. Dado que las imágenes son proyecciones 2D del mundo 3D, la información no está disponible directamente y debe ser recuperada. Existen técnicas desarrolladas en otras áreas que son utilizadas para recuperar información desde imágenes. Algunas de ellas son el Procesamiento de Imágenes (generalmente en las primeras etapas de un sistema de visión por computadora, donde se usa para mejorar información particular y suprimir ruido) [Gon92][Bax94][Hus91][Jai89], Gráficos por Computadora (donde se generan imágenes a partir de primitivas geométricas), Reconocimiento de Patrones (el cual clasifica datos numéricos y simbólicos) [Sch92], etc.

Un tipo de problema particular que interesa es el de los procesos de clasificación y reconocimiento de objetos que provienen de una línea industrial (de producción o packaging) [Law92]. La incorporación de computadoras en este tipo de problemas tiende a incrementar tanto la cantidad como la calidad de los productos obtenidos y a la vez reducir los costos. Estos sistemas llevan a cabo tareas simples como el procesamiento de imágenes obtenidas bajo condiciones de iluminación controladas y desde un punto de vista fijo. Además, deben operar en tiempo real, lo que implica el uso de hardware especializado y algoritmos robustos para reducir la posibilidad de fallas, y deben ser flexibles para poder adaptarse rápidamente a cambios en el proceso de producción y a los requisitos de inspección [Lev90]. Esta restricción temporal naturalmente lleva a pensar en una resolución del problema utilizando procesamiento paralelo, para cumplir adecuadamente con los requerimientos.

En el presente trabajo se tomó como modelo para la clasificación una cinta transportadora sobre la cual viajan los objetos. En un punto del recorrido se encuentra una video cámara que toma imágenes de los mismos, las cuales son recibidas por una computadora. A partir de estas imágenes la aplicación clasifica al objeto y toma decisiones que eventualmente se traducen en señales que pueden derivarlo a distintos sectores según sus características (calidad, color, tamaño, etc.) o descartarlo.

La toma de decisiones siempre requiere conocimiento de la aplicación u objetivo: en cada etapa de un sistema de visión se deben tomar decisiones. El énfasis está puesto en maximizar la operación automática en cada etapa, y los sistemas deben usar conocimiento (implícito y explícito) para realizar esto. La eficacia y eficiencia del sistema está gobernada usualmente por la calidad del conocimiento empleado.

Una solución informática carecería de los problemas presentados por los métodos mecánicos, electrónicos o manuales, ya que permite analizar un amplio espectro de características, es fácilmente adaptable a distintas clases de objetos y no cuenta con las limitaciones biológicas de los humanos. Sin embargo, las aplicaciones de reconocimiento de objetos existentes cuentan con limitaciones de performance que restringen su uso a casos muy específicos donde los tiempos de respuesta no son determinantes o a proyectos de investigación donde se cuenta con máquinas especializadas de gran velocidad de cómputo.

Intentando superar estas limitaciones, este trabajo tiene como objetivo principal el desarrollo de algoritmos de clasificación y reconocimiento en paralelo, que permitan llegar a resultados en tiempos aceptables utilizando una red de procesadores heterogéneos (por ejemplo con Pcs convencionales).

2. Visión por computadora

La visión por computadora desarrolla teorías y algoritmos básicos por los cuales la información útil acerca del mundo puede ser automáticamente extraída y analizada a partir de imágenes. Para entender mejor los problemas y las metas que constituyen el propósito del procesamiento de imágenes es necesario comprender la complejidad que involucra nuestro propio sistema humano de visión.

Aunque la captación visual, el reconocimiento de objetos, y el análisis de la información recogida parece un proceso instantáneo para el ser humano, el cual lo realiza en forma "natural", nuestro sistema de visión aún no ha sido comprendido en su totalidad. Involucra sensores físicos que captan la luz reflejada por el objeto en cuestión (creando la imagen original), filtros que limpian la imagen (perfeccionándola), sensores que captan y analizan particularidades de la imagen para que, luego de un reconocimiento y clasificación de la misma, se puedan tomar decisiones en base a la información acumulada. Así planteado, se puede apreciar la complejidad del aparentemente sencillo proceso de ver, el cual comprende una cantidad numerosa de subprocesos, todos realizados en muy poco tiempo.

El propósito y la estructura lógica de un sistema de visión por computadora es esencialmente el mismo que el de un sistema de visión humano. A partir de una imagen captada por un sensor, se realizan todos los procesos y análisis necesarios para llegar al reconocimiento de la misma y de los objetos que la componen. En el diseño de un sistema de visión existen diversas consideraciones:

- qué tipo de información se intenta extraer de la imagen?
- cuál es la estructura de esta información en la imagen?
- qué conocimiento a priori se necesita extraer de esa información?
- qué tipos de procesos computacionales se requieren?
- cuáles son las estructuras para conocimiento y representación de datos requeridas?

En el procesamiento requerido se encuentran cuatro aspectos principales:

- Preprocesamiento de los datos de la imagen
- Detección de rasgos del objeto
- Transformación de datos icónicos en simbólicos
- Interpretación de la escena

Cada una de estas tareas requiere una representación de datos distinta, como así también requerimientos computacionales diferentes. Esto hace surgir el siguiente interrogante: Qué arquitectura se necesita para llevar a cabo las operaciones?.

Muchas arquitecturas tienen desarrollados computadores en pipeline, lo que provee una concurrencia limitada de operaciones, aunque a una buena tasa de transferencia de datos (throughput). Otras cuentan con una malla conectada de arreglos porque su mapeo de imágenes es eficiente, o aumentan la arquitectura en malla con árboles o pirámides porque proveen la jerarquía de operaciones que, se cree, involucra el sistema de visión biológico. Algunas arquitecturas son desarrolladas con máquinas paralelas más generales basadas en memoria compartida o hipercubos conectados.

Los problemas que se presentan en un sistema de visión ocurren porque las unidades de observación no son unidades de análisis. La unidad de una imagen digital observada es el pixel, que posee propiedades de posición y valor; pero el hecho de que se conozca la posición y el valor de cualquier pixel no aporta información sobre el reconocimiento de un objeto, la descripción de su forma, su orientación, la medida de cualquier distancia en el objeto, su grado de defectuosidad o simplemente información sobre qué pixels son parte del objeto en cuestión.

Reconocimiento de objetos

El proceso que permite clasificar los objetos de una imagen es llamado reconocimiento e inspección computacional, e involucra por su complejidad una variedad de etapas que sucesivamente transforman datos icónicos para reconocer información. La detección de objetos en tiempo real, contando sólo con la información procesada de la imagen, es una tarea problemática pues la capacidad de reconocimiento y clasificación es función de numerosos factores. Entre ellos se encuentran:

- la similitud entre objetos del conjunto analizado con los del que se quiere detectar
- los distintos ángulos posibles usados en la visualización de los objetos
- el tipo de sensores involucrados
- la cantidad de distorsión y ruido
- los algoritmos y procesos empleados
- la arquitectura de los computadores utilizados
- las restricciones específicas de tiempo real
- los niveles de confidencialidad requeridos para tomar la decisión final

Un sistema reconocedor de objetos "ideal" puede suponer que el conjunto de objetos es observado por un sensor perfecto, sin transmisión de ruido ni distorsión. Un enfoque más realista supone algo distinto. La imagen es creada por un proceso matemático aplicado a los datos sensados, y luego se la limpia o restaura para clasificar los rasgos que serán necesarios para procesos posteriores. La segmentación la divide en piezas mutuamente exclusivas para facilitar el reconocimiento. Se seleccionan datos útiles y se registran aspectos relevantes aportados por los sensores (altitud, velocidad, etc.) para poder llevar a cabo la clasificación. Los objetos segmentados se chequean contra una clase predeterminada para saber si los objetos buscados están presentes, con lo que se completa el decisivo proceso de detección.

Las metodologías de reconocimiento más difundidas generalmente constan de las siguientes etapas:

1. *Formación de la imagen*: es la representación digital de los datos captados por el sensor. Incluye la determinación del muestreo (sampling) y la cuantización de niveles de gris, los cuales definen la resolución de la imagen.

2. *Acondicionamiento*: se trata de realzar y restaurar la imagen original para obtener una "mejor", entendiéndose por "mejor" lo que el observador considere como calidad en la imagen. Se pueden usar técnicas de realzado del contraste, smoothing y sharpening. La restauración consiste en la reconstrucción de una imagen degradada conociendo a priori la causa de la degradación (movimiento, foco, ruido, etc.).

3. *Labeling (etiquetado) de eventos*: se basa en el modelo que define al patrón informativo con una estructura de arreglo de eventos (o regiones), donde cada evento es un conjunto conectado de pixels. El labeling determina a qué clase de evento pertenece cada pixel, colocando igual etiqueta a los pixels de un mismo evento. Como ejemplos de labeling se pueden mencionar el thresholding (con valores de pixels altos o bajos), la detección de bordes, búsqueda de esquinas, etc.

4. *Agrupamiento de eventos*: identifica los eventos, para lo cual puede colectar simultáneamente o identificar conjuntos de pixels muy conectados que estén en una misma clase de evento. Puede definirse como una operación de componentes conectadas (si las etiquetas son simbólicas), una operación de segmentación (si las etiquetas son en niveles de grises), o un enlazamiento de bordes (si las etiquetas son pasos de bordes). Luego de esta etapa, las entidades de interés ya no son pixels sino conjuntos de pixels.

5. *Extracción de propiedades*: asigna nuevas propiedades al conjunto de entidades que genera el agrupamiento. Algunas de ellas son el centro, el área, la orientación y los momentos. También puede determinar relaciones entre grupos de pixels, como por ejemplo si se tocan, si son cerrados, si un grupo está dentro de otro, etc.

6. *Matching*: los eventos o regiones de la imagen son identificados y medidos a través de los procesos anteriores, pero para poder asignarles un significado es necesario realizar una organización conceptual que permita identificar un conjunto específico como una instancia imaginada de algún objeto previamente conocido. El matching determina la interpretación de algunos conjuntos de eventos relacionados, asociando esos eventos con algún objeto. Las figuras simples se corresponderán con eventos primitivos y la propiedad de medición desde un evento primitivo con frecuencia será adecuada para permitir el reconocimiento de la figura. Las figuras complejas se corresponderán con conjuntos de eventos primitivos. En este caso, para el reconocimiento se utilizará un vector de propiedades de cada evento observado y las relaciones entre los mismos.

Excepto la formación de la imagen, el resto de las etapas pueden ser aplicadas varias veces en distintos niveles de procesamiento.

El manejo de ambientes no restringidos es una de las dificultades actuales, pues los algoritmos de visión por computadora y reconocimiento existentes son especializados y no desarrollan más que algunos de los pasos de transformación necesarios.

3. Descripción del problema y solución secuencial

Inicialmente se tomó como ejemplo de procesos industriales el caso de cintas transportadoras de huevos, ya que la regularidad de estos objetos facilita los procesos de clasificación. Luego se consideraron casos mas complejos como frutas, botellas, cerámicas, etc. La aplicación clasifica a los objetos y toma decisiones que pueden traducirse como señales que los deriven a distintos sectores de la línea de producción según sus características o los descarten. Para la clasificación se toman en cuenta diversas características de los objetos, como su área, color, diámetro máximo, perímetro, centroide y la presencia de defectos sobre su superficie (manchas, decoloraciones, etc.).

Como ya se mencionó, el objetivo es automatizar la clasificación de los objetos utilizando algoritmos distribuidos en redes de máquinas heterogéneas. Para ello, se describe en primer lugar la resolución secuencial del problema y luego su paralelización.

Con el fin de concentrar la atención específicamente en el desarrollo de los algoritmos de reconocimiento distribuidos, se trabajó sobre imágenes true color bidimensionales de los objetos en cuestión, dejando para una futura extensión la etapa de adquisición de las mismas por medio de una videocámara.

Para obtener las imágenes utilizadas, en un primer momento se fotografiaron huevos de gallina sobre un fondo oscuro con luz natural. Una vez reveladas las fotografías, se scannearon utilizando un scanner Genius 9000, con una resolución de 300 dpi.

Al efectuar las primeras pruebas con las imágenes, se observó que la iluminación natural provocaba reflejos y sombras sobre la superficie de los objetos, lo que impedía la exacta determinación de su color. Por esta razón, en una segunda etapa se construyó un set con iluminación artificial desde distintos ángulos, y las fotografías obtenidas mostraron una sensible mejora. También se obtuvieron muestras de frutas (limones, naranjas y tomates) y de cerámicas (mosaicos y azulejos). Finalmente, las imágenes scanneadas fueron almacenadas en formato JPEG para reducir el espacio de almacenamiento. A continuación se describen los algoritmos implementados.

3.1. *Threshold*

El primer algoritmo implementado fue el threshold, utilizado para diferenciar el fondo de la imagen de los objetos en cuestión. Es una operación de labeling sobre una imagen en escala de grises o colores. Distingue pixels con un valor alto de gris o color de los que tienen un valor bajo. Es uno de los métodos más utilizados para extraer una figura o un rasgo de interés particular de una imagen. El operador binario de threshold produce una imagen en blanco y negro asignando (por ejemplo) el valor correspondiente al negro a pixels con valor más alto que el elegido como umbral, y blanco a los que tienen un valor más bajo.

Un punto importante en esta operación es la adecuada elección del valor del umbral, ya que de lo contrario las etapas posteriores pueden fracasar. Existen distintos métodos para elegir automáticamente el valor ideal de threshold; uno de ellos se basa en el histograma de la imagen. Para encontrar un valor que separe el fondo oscuro del objeto brilloso (o viceversa) de una manera óptima, sería deseable conocer la distribución de los pixels oscuros y la de los brillosos. Así, el valor de corte podría determinarse como ese valor de separación por el cual la probabilidad de etiquetar mal un pixel del fondo sea la misma que la de etiquetar mal un pixel del objeto. La dificultad radica en que la distribución independiente de pixels oscuros y brillosos generalmente no es conocida en un primer momento.

La idea del algoritmo es muy simple y consiste en recorrer cada pixel de la imagen cambiándolo por el valor 255 (blanco) si su valor se encuentra por encima del valor preestablecido de threshold (VDT), o por el valor 0 (negro) en caso contrario. Nótese que el algoritmo depende del VDT. En base a los resultados obtenidos con las primeras imágenes procesadas, se notó que el VDT ideal depende de las condiciones de iluminación y del fondo de la imagen procesada, por lo que un mismo valor sirve para grupos de imágenes obtenidas en las mismas condiciones. Si el VDT es mal determinado, el algoritmo puede incurrir en fallas tales como tomar pixels del fondo como parte de los objetos o como objetos separados (en caso que el VDT sea menor al ideal), o confundir pixels de los objetos con el fondo.

3.2. *Labeling de componentes conectadas. Primera versión*

El segundo algoritmo implementado fue el labeling, consistente en el etiquetado de las componentes conectadas en una imagen binaria, seguido de mediciones sobre las regiones encontradas. Los algoritmos para labeling de componentes conectadas agrupan juntos todos los pixels pertenecientes a la misma región y les asignan una etiqueta única (es una operación de agrupamiento). La región es una unidad más compleja que el pixel y tiene un conjunto de propiedades más completo (forma, posición, estadísticas de nivel de gris, etc.), por lo que para cada región se puede construir una n-tupla de esas propiedades. Una forma de reconocer objetos diferentes, defectuosos o características es distinguir entre las regiones en base a sus propiedades.

La velocidad del algoritmo que ejecuta esta operación de etiquetado es crítica, y existen distintos variantes para realizar esta etapa. Normalmente procesan una fila de la imagen a la vez y asignan nuevas etiquetas al primer nivel de cada componente, intentando propagar la etiqueta de un pixel a sus vecinos de la derecha o abajo.

Existe un algoritmo, propuesto por Haralick, que se caracteriza por no utilizar un almacenamiento auxiliar para producir la imagen etiquetada. Cuenta con un paso de inicialización más una secuencia de propagación de etiquetas en forma top-down, seguida de una propagación bottom-up iterando hasta que no ocurran cambios en las etiquetas.

Otra posibilidad es utilizar el denominado "algoritmo cíclico", basado en el algoritmo clásico de componentes conectadas para grafos descripto por Rosenfeld y Pfaltz. Realiza sólo dos pasadas por la imagen pero requiere una extensa tabla global para almacenar equivalencias. La primera pasada ejecuta la propagación de etiquetas. Cuando se presenta una situación en la que dos etiquetas distintas pueden ser propagadas al mismo pixel, se propaga la más pequeña y cada equivalencia encontrada se agrega a la tabla. Cada entrada en la tabla de equivalencias es un par ordenado (etiqueta, etiqueta equivalente).

Luego de la primera pasada se encuentran las 'clases de equivalencias' haciendo la clausura transitiva del conjunto de equivalencias de la tabla. Cada clase de equivalencia es asignada a una única etiqueta, frecuentemente la menor (o la más antigua) de la clase. La segunda pasada asigna a cada pixel la etiqueta de la clase de equivalencia correspondiente a la etiqueta que se le asignó al pixel en la primera pasada. El principal inconveniente de este enfoque radica en la tabla de equivalencias global, ya que en imágenes grandes con muchas regiones se torna demasiado extensa.

La idea inicial fue procesar la imagen en base al threshold, pero pronto se advirtió que con los resultados de éste no era suficiente, ya que la imagen generalmente posee imperfecciones (producidas en general por efecto de la iluminación) que se consideraban como objetos. Por otra parte, implementar un algoritmo de labeling permitiría la clasificación de varios objetos en una misma imagen, así como el procesamiento de marcas de control para, por ejemplo, determinar el tamaño de los objetos en una unidad de medida standard.

El operador de labeling también genera una tabla de regiones que indica la cantidad de pixels de cada una de ellas. Analizando dicha tabla, podemos obtener información acerca de cuál es el objeto de mayor tamaño contenido en la imagen (el objeto de interés), cuáles son los objetos muy pequeños que representan ruido o imperfecciones en la misma, y objetos de tamaño preestablecido que en nuestro caso representan marcas de control.

La primera implementación realizaba dos pasadas por cada línea horizontal de la imagen. En la primera pasada de izquierda a derecha etiquetaba los pixels de acuerdo a sus vecinos superior y anterior, y en la segunda pasada que se realizaba de derecha a izquierda resolvía las equivalencias que se generaban en esta línea observando únicamente el pixel que quedaba a la derecha, según el siguiente algoritmo:

```
Procedure Labeling_V1
 // La primera linea es un caso especial ( no tiene arriba )
 Para c/pixel de la linea  que no es fondo // distinto de negro
    si el pixel anterior tiene etiqueta
       etiqueta = etiqueta del pixel anterior
    si no
       etiqueta = etiqueta nueva
    fin si
 fin para
```

```
// Ahora procesamos el resto de la imagen
Para cada linea del resto de la imagen
    // Primero de izquierda a derecha
    Para c/pixel de la linea que no es fondo // distinto de negro
        si pixel de arriba y el pixel de la izquierda estan etiquetados
            etiqueta = mín ( etiqueta de arriba, etiqueta de la izquierda)
        sino si el pixel de la izquierda tiene etiqueta
            etiqueta = etiqueta de la izquierda
        sino si el pixel de arriba tiene etiqueta
            etiqueta = etiqueta de arriba
        sino
            etiqueta = etiqueta nueva
        fin si
    fin para

    // Ahora de derecha a izquierda
    Para c/pixel de la linea que no es fondo // distinto de negro
        si el pixel de la izquierda tiene etiqueta
            etiqueta = etiqueta de la izquierda
        fin si
    fin para
fin para
fin Labeling_V1
```

Este algoritmo funcionó muy bien con objetos de forma circular o poligonal, pero no con regiones del tipo de herraduras con la abertura en la parte superior, pues las equivalencias detectadas en una línea no se propagaban hacia las superiores.

3.3. *Labeling de componentes conectadas. Segunda versión*

Para resolver los problemas presentados en la primera versión se decidió realizar una nueva implementación que realizara una primera pasada sobre la imagen, línea a línea y en un solo sentido.

Las equivalencias que se presentan se almacenan en una tabla que se genera durante la primera pasada. Luego se realiza una segunda pasada en el mismo sentido que la anterior (aunque esto no es importante) durante la cual, para cada pixel, se cambia su etiqueta por la que le corresponda a la etiqueta que tenia en la tabla de equivalencias (si la tuviera).

El problema de este algoritmo es el mantenimiento de la tabla de equivalencias, ya que al ir agregando equivalencias se genera un grafo que determina las equivalencias de cada etiqueta. Para conocer estas equivalencias es necesario realizar la clausura transitiva de dicho grafo, lo que no resulta una tarea trivial y dependiendo de la forma de la imagen se puede llegar a degradar la performance del algoritmo. De todas maneras, funciona correctamente para cualquier forma geométrica.

```
Procedure Labeling_V2;
// La primera linea es un caso especial ( no tiene arriba )
Para c/pixel de la linea  que no es fondo // distinto de negro
    si el pixel anterior tiene etiqueta
        etiqueta = etiqueta del pixel anterior
    si no
        etiqueta = etiqueta nueva
    fin si
fin para
```

```
// Ahora procesamos el resto de la imagen
Para cada linea del resto de la imagen
    Para c/pixel de la linea que no es fondo // distinto de negro
        si pixel de arriba y el pixel de la izquierda estan etiquetados
            etiqueta = mín( etiqueta de arriba, etiqueta de la izquierda)
            si etiqueta arriba <> a la de la izquierda
              //agrego equivalencia a la tabla
              etiq. pixel equivale a max( etiqueta de arriba, etiqueta de la zquierda )
            fin si
        sino si el pixel de la izquierda tiene etiqueta
            etiqueta = etiqueta de la izquierda
        sino si el pixel de arriba tiene etiqueta
            etiqueta = etiqueta de arriba
        sino
            etiqueta = etiqueta nueva
        fin si
    fin para
fin para

// Ahora realizamos una segunda pasada resolviendo equivalencias
Para cada linea de la imagen
    Para c/pixel de la linea que no es fondo // distinto de negro
        si la etiqueta del pixel tiene equivalencia
            etiqueta = equivalencia
            // equivalencia es el resultado de la clausura transitiva
        fin si
    fin para
fin para
fin Labeling_V2
```

3.4. *Cálculo del color promedio*

Luego de identificar los pixels que forman el objeto a clasificar (que es la región de mayor tamaño resultante del labeling) se calcula el color promedio simplemente sumando los colores de estos pixels y dividiéndolo por la cantidad. Para llegar a un resultado satisfactorio es indispensable que el objeto en cuestión sea en forma mayoritaria de un color uniforme. Además, la imagen debe haber sido adquirida en condiciones de iluminación que no alteren las características del mismo.

3.5. *Identificación de manchas*

Uno de los objetivos de la aplicación es determinar la presencia de defectos en el objeto a clasificar. Con este fin, se desarrolló un algoritmo que encuentra estos defectos o manchas dentro del objeto principal identificado en el labeling.

El algoritmo funciona de manera similar al threshold, solo que para modificar el valor de cada pixel por blanco o negro no se utiliza uno sino dos valores: si el valor del pixel está dentro del rango determinado por ellos se le asigna 0 (negro) ya que se trata del objeto, y en caso contrario se le asigna 255 (blanco) pues se trataría de una mancha. Los dos valores mencionados se determinan en base al color promedio del objeto y a un parámetro que indica la variación de color considerada normal.

Al analizar los resultados obtenidos con este algoritmo se notó que si la imagen presentaba variaciones de tonalidad debidas a diferencias de iluminación sobre la superficie del objeto, se detectaban como manchas a zonas que en realidad eran poco o muy iluminadas. Para solucionar esto, se implementó otro algoritmo que asume que el color predominante en el objeto presenta diferencias de intensidad. Así, para determinar si un pixel es o no un defecto del objeto se controla si su color es similar al color promedio encontrado, aceptando una variación lineal en las tres componentes (red, green y blue), en cuyo caso, a pesar de no ser del mismo color es considerado normal (no perteneciente a un defecto del objeto). Para ello se realiza el siguiente calculo:

1°- Normalización de las componentes RGB del color promedio:
$Componente_i\ Normalizada = Componente_i\ Promedio * 100 / 255 \quad \forall i \in \{red, green, blue\}$

2°- Para cada pixel del objeto, se calcula:
$Componente_i\ Diferencia = (Componente_i\ del\ pixel * 100 / 255) - Componente_i\ Normalizada$

3°- En el caso ideal se esperaría que los tres valores diferencia fueran equivalentes para los pixels del color del objeto, y distintos para los que son defectos. Como esto no se cumple siempre, se admite un margen de tolerancia en la variación entre estos valores diferencia.

El resultado es una imagen binaria con las manchas encontradas en el objeto en color blanco. Sobre ella se realiza un labeling con el fin de calcular la cantidad de manchas o defectos y el tamaño de los mismos.

3.6. *Detección de los bordes*

Analizando el borde o perímetro del objeto, es posible encontrar datos acerca del mismo, como por ejemplo su forma, sus diámetros, etc. Por este motivo, se creyó útil incorporar esta facilidad a la aplicación.

Se considera que un pixel de una región pertenece al borde de la misma si algún pixel vecino (utilizando conectividad 4) no pertenece a la región. A medida que se encuentran los pixels se guardan sus coordenadas en una lista para poder procesarlas más tarde rápidamente sin necesidad de recorrer la imagen.

El algoritmo encuentra tanto bordes externos como internos en el caso de que la región presente orificios, lo que se debe tener en cuenta al utilizar las coordenadas de estos pixels para el cálculo de los momentos.

3.7. *Determinación del centroide o centro de masa*

Uno de los momentos de primer orden de una región es el centroide o centro de masa de la misma. En el caso de círculos o cuadrados, coincide con el centro geométrico. El interés por calcular este valor se debe a que basándose en él y en el borde se pueden determinar características interesantes de los objetos, y también a que si existieran en la imagen marcas para facilitar el cálculo de distancias sobre las mismas, dichas distancias se miden desde el centro de las marcas, por lo que éste debe ser determinado. El cálculo es el siguiente:

$$\text{Centroide de } R = C_{xy} = \text{Suma } (x, y) / \text{Area } (R)$$
$$\forall (x,y) \in R$$

3.8. *Cálculo de momentos de primer orden*

Existen dos momentos de primer orden que permiten identificar la forma general de un objeto. Estos son:

Distancia promedio desde el centro al perímetro:

$\mu_r = (1/K) * S (| (x,y) - C_{xy} |), \quad \forall (x,y) \in$ perímetro, y K el número de pixels del mismo.

Desvío standard de μ_r:

$s_r^2 = (1/K) * S (| (x,y) - C_{xy} | - \mu_r)^2$

4. Algoritmos y lenguajes paralelos

Los factores que afectan la performance de un algoritmo en una arquitectura particular dependen del grado de paralelismo y de la sobrecarga por scheduling y sincronización de tareas. Para el paralelismo a nivel de tareas, cuando más fino es el grano de paralelismo mayor es el costo de scheduling y sincronización, lo cual lleva a que no se pueda conseguir un incremento lineal en la velocidad por un incremento en el número de procesadores. La elección de un algoritmo para resolver un problema particular está fuertemente influenciada por la arquitectura de hardware y las herramientas de software disponibles [Cof92][Hee91][Hwa93][Lei92][Mor94].

Para la elección del modelo de paralelismo a utilizar (de datos, de tareas o sistólico), se debe tener en cuenta que no cualquier tipo de paralelismo resolverá un problema dado en forma eficiente. Con frecuencia los algoritmos se desarrollan sin referencias a una arquitectura particular, y usan más de un modelo de paralelismo, lo que los hace difíciles de implementar.

El paradigma de paralelismo de datos es explícitamente sincronizado y se mapea al modelo de programación SIMD. Aunque es un modelo óptimo, por ejemplo, para el procesamiento de imágenes a bajo nivel, no es obvio cómo representar o implementar tareas de razonamiento de alto nivel. El paralelismo a nivel de tareas requiere que el algoritmo sea particionado en subtareas, y que éstas y los datos sean distribuidos entre los procesadores, por lo que el sistema debe estar configurado para permitir la comunicación y sincronización entre los procesadores. En el modelo sistólico, el algoritmo es particionado temporalmente y cada etapa computa un resultado parcial y lo pasa a la siguiente.

Para que un lenguaje sea considerado paralelo debe soportar alguno de los conceptos de paralelismo y comunicación. El paralelismo puede ser a nivel de procesos (Ada), objetos (Emeral), sentencias (Occam) [CSA90][Hoa85], expresiones (lenguajes funcionales), o cláusulas (Parlog). La comunicación puede ser punto a punto o broadcast.

Parallel Virtual Machine (PVM)

Es un conjunto integrado de herramientas de software y librerías que emulan en forma flexible un framework para computaciones concurrentes heterogéneas de propósito general, sobre computadoras interconectadas de diversas arquitecturas [PVM96]. El objetivo principal del sistema PVM es permitir que computaciones concurrentes o paralelas sean ejecutadas sobre una colección de computadoras. Los principios en los que se basa PVM son:

- Pool de computadoras configurado por el usuario (incluso puede ser modificado durante la ejecución agregando o quitando máquinas.
- Acceso transparente al hardware (aunque se puede elegir explotar las capacidades de máquinas específicas ubicando ciertas tareas en las computadoras más apropiadas).
- Computación basada en procesos. La unidad de procesamiento es la tarea, y múltiples tareas pueden ejecutarse en un solo procesador.
- Modelo de pasaje de mensajes explícito. Las tareas cooperan enviando y recibiendo mensajes explícitos una de la otra.
- Soporte de heterogeneidad (en términos de máquinas, redes y aplicaciones)
- Soporte de multiprocesadores. Usa las facilidades nativas para el pasaje de mensajes en multiprocesadores para sacar ventajas del hardware específico de los mismos.

El modelo computacional está basado en la noción de que una aplicación consta de varias tareas, donde cada una de ellas es responsable de una parte del trabajo. Soporta paralelismo de tareas y de datos (incluso mezclados). Dependiendo de las funciones, las tareas pueden ejecutarse en paralelo y pueden necesitar sincronizarse e intercambiar datos, a través del envío de mensajes.

El sistema PVM actualmente soporta los lenguajes C, C++ y Fortran. Estos lenguajes se han elegido como consecuencia de que la mayoría de las aplicaciones se escriben usando C o Fortran y a las nuevas aplicaciones basadas en lenguajes y metodologías orientados a objetos. La interfase de la librería de usuario para C y C++ está implementada como funciones.

El paradigma general de programación de una aplicación que utiliza PVM es el siguiente: el usuario escribe uno o más programas secuenciales en los lenguajes soportados que contienen llamadas a la librería. Cada programa se considera una tarea que forma parte de la aplicación. Para la ejecución, generalmente el usuario dispara una instancia de la tarea "maestra" desde una máquina incluida en el pool, la cual irá ejecutando a su vez otras tareas, resultando un conjunto de procesos que realizan cálculos locales e intercambian mensajes con otros para resolver el problema.

5. Paralelización de los algoritmos

Para realizar el procesamiento en paralelo de las imágenes se utilizó un modelo de "master/slave". Existe un proceso maestro que es el responsable de repartir el procesamiento entre un número determinado de procesos esclavos o clientes, y de coordinar los cálculos de los mismos. Los clientes son los encargados de llevar a cabo el procesamiento propiamente dicho.

Hay paralelización a nivel de datos: la imagen se particiona en subimágenes que son enviadas a los clientes (todos idénticos) encargados del procesamiento. Algunos algoritmos son fácilmente paralelizables, mientras otros requieren un cuidado especial que involucra la participación del master en el procesamiento para coordinar a los clientes. A continuación se analiza cada uno de los algoritmos desarrollados.

5.1. *Threshold*

No necesita ninguna consideración especial, ya que solamente se analiza cada pixel de la región comparándolo con el valor límite, y por lo tanto el resultado obtenido es el mismo independientemente del número de clientes que se utilicen.

5.2. *Labeling*

Presenta dificultades en la paralelización debido principalmente a que las regiones pueden quedar divididas en dos o más subimágenes, y también a que las etiquetas deben ser únicas. Para resolver estos inconvenientes, se decidió dividir a las dos pasadas que se realizan en el labeling descripto anteriormente.

Cada cliente realiza la primera pasada sobre la subimagen que tiene asignada del mismo modo en que se hace en el algorimo convencional. Luego, le envia al proceso master su tabla de labeling local. Para resolver el conflicto que se genera cuando una región queda dividida en dos o más subimágenes, las lineas correspondientes al punto de división de la imagen se envían a dos clientes, a uno como última linea de una subimagen y a otro como primera línea.

Los procesos clientes envían al master también estas dos lineas especiales con las etiquetas que les asignaron. El master concatena las tablas de labeling parciales que le envian los clientes en una tabla global y simultáneamente revisa las lineas superpuestas; en caso de generarse un conflicto (los dos clientes involucrados etiquetaron los pixels de esa linea con etiquetas distintas) agrega la equivalencia correspondiente en la tabla global.

Luego de procesar esta información, el master envía a los clientes la porción de tabla de labeling que le correspondia a cada uno, pero que ahora puede tener nuevas equivalencias. Los clientes la reciben y realizan su segunda pasada del labeling basándose en esta tabla modificada. Cuando se forma la tabla global, se soluciona también el problema de dos regiones distintas que tengan la misma etiqueta, ya que en la tabla global se modifican las etiquetas de las parciales sumándoles un número de manera de obtener etiquetas distintas.

En este algoritmo, el proceso master realiza una parte del procesamiento: concretamente unifica las tablas de labeling y resuelve los conflictos en las líneas de imagen solapadas. Esto puede resultar ineficiente en casos en que el número de procesadores es muy grande (por ejemplo si hay un procesador por linea de imagen), ya que el master estaría procesando muchas lineas, con lo que se perderían las ventajas del paralelismo. Como esta aplicación fue pensada para ejecutarse en una red local con un número limitado de máquinas convencionales, esto no es un problema (por ejemplo si se cuenta con 5 máquinas y una imagen de 200 líneas el master sólo procesaría cinco lineas, o sea, el 2.5% de la imagen). A su vez, esto disminuye el uso de la red de comunicaciones ya que de otra manera los clientes deberian comunicarse entre sí y con otro proceso que provea etiquetas únicas.

5.3. *Cálculo del valor promedio*

Para el cálculo distribuido del color promedio se usó el algoritmo descripto anteriormente, con la diferencia de que ahora cada cliente envía al master su resultado parcial, y éste calcula el promedio entre todos los clientes, teniendo en cuenta que elste promedio debe ser pesado con la cantidad de pixels que tuvo en cuenta cada proceso.

5.4. *Detección de los bordes*

Para realizar la búsqueda de los pixels que forman el perímetro del objeto en paralelo, se debe tener en cuenta que la imagen se encuentra dividida. Esto implica que no se puede tomar la convención de que los pixels de la región que estén situados sobre la primera y la última línea pertenecen al perímetro de la región, ya que puede tratarse de una partición intermedia de la imagen.

Para resolver este inconveniente es necesario que los procesos sepan qué porción de la imagen están procesando. Estas porciones pueden ser: la primera (lo que significa que la primera línea de su subimagen es límite de la imagen completa), una intermedia (ni la primera ni la última línea son límite de la imagen completa), o la última (con lo cual la última línea es límite de la imagen completa). También se debe tener en cuenta que las líneas sobre las cuales se divide la imagen son procesadas dos veces, y es necesario, en caso de que el borde de la región esté sobre ellas, evitar contarlo dos veces.

Para esto se desarrolló un algoritmo similar al convencional, pero que tiene en cuenta todos estos casos. Este algoritmo soluciona los problemas mencionados anteriormente y por lo tanto permite identificar los pixels que pertenecen al perímetro de la región sin duplicados y evitando tomar como parte del borde a los pixels de los bordes de las particiones intermedias de la imagen original si es que no lo son.

6. Ambiente de desarrollo e implementación

Para la implementación se tuvo en cuenta que ésta fuera clara y fácilmente extensible, y además que PVM estaba disponible para ser utilizado con C y Fortran. Por este motivo, fue desarrollada en C++ (utilizando PVM y en ambiente Linux), ya que provee posibilidades de programación orientada a objetos [Rum91], lo que facilita el desarrollo de un código claro, modular y extensible, y además es posible desde C++ acceder a las rutinas PVM que están desarrolladas en C.

Como se mencionó anteriormente, la aplicación está orientada a convertirse en un proceso que se aplica en tiempo real sobre las imágenes que son obtenidas por la videocámara, y sus resultados son enviados a dispositivos que deciden el destino de los objetos de acuerdo a las características de cada uno de ellos. Sin embargo, para el desarrollo fue necesario utilizar un ambiente gráfico para poder observar los resultados de las imágenes en cada paso de procesamiento de manera de poder determinar la corrección de los algoritmos.

El sistema gráfico X-Windows de Linux, desarrollado sobre una arquitectura cliente-servidor [Uma93], permite, en forma transparente, ver datos en pantalla generados en cualquier máquina de la red. Si bien es simple el manejo de elementos de interfase utilizando la API del sistema X-Windows, no es sencillo mostrar imágenes true color como las manejadas en la aplicación.

Para simplificar el desarrollo se decidió utilizar una aplicación de procesamiento de gráficos simple como la librería Gimp, cuya ventaja es la posibilidad de agregar procesos plug-in. Por lo tanto, se decidió desarrollar la aplicación gráfica como un plug-in de Gimp y de esta manera poder utilizar su funcionalidad para mostrar imágenes en forma transparente. El Gimp permite abrir una imagen en varios formatos (Jpeg, Tiff, Targa, Gif, etc), mostrarla, y aplicarle una serie de procesos implementados en forma de plug-in. Estos procesos se configuran agregándolos a un archivo (.gimprc), y Gimp permite al usuario seleccionar uno de los procesos de la lista para aplicarlo a la imagen cargada. Una vez elegido un proceso, Gimp lo ejecuta y le envía la imagen a procesar por medio de pipes. El proceso corre de forma independiente de Gimp, y puede comunicarse con éste por medio de una API sencilla que permite mostrar nuevas imágenes, diálogos para el ingreso de datos, etc.

La implementación consta de dos aplicaciones: una gráfica (como se expresó, básicamente para el desarrollo y testeo del sistema) y otra de tipo "batch" (que es la que puede ser utilizada en combinación con las etapas de adquisición y desvío de los objetos al lugar adecuado).

La aplicación gráfica consiste en un plug-in de Gimp, que al ser ejecutado sobre una imagen previamente leída presenta un diálogo que permite especificar los parámetros de procesamiento. Estos parámetros son: la cantidad de procesos a utilizar, el valor del umbral, el tamaño de los defectos a tener en cuenta en relación con el objeto, la diferencia de color de una región con respecto al color promedio para ser considerada defecto, si se desean calcular los momentos y si se desea mostrar los resultados parciales (solo para debugging). Luego se realiza el procesamiento, y al finalizar éste se muestra un diálogo con los resultados: superficie (cantidad de pixels) del objeto principal, color promedio del objeto, diámetro máximo, valores (opcionales) de los momentos μ_r y s_r^2, la cantidad de manchas (defectos), superficie de los defectos, centroide del objeto, centro de marcas de referencia (si es que se encontraron) y tiempo del procesamiento.

Por otra parte se realizó la aplicación "batch", que fue útil para realizar mediciones de performance. El proceso es muy sencillo y permite el pasaje como parámetros en la línea de comandos de los mismos valores que se utilizan en la aplicación gráfica. Al finalizar el procesamiento se ven en pantalla los resultados del mismo. Para levantar imágenes con formato Jpeg se utilizó una librería provista en la distribución de Linux (libjpeg).

En la implementación se definieron clases para abstraer los principales objetos intervinientes en la aplicación, los cuales son imágenes RGB compuestas por pixels, y algoritmos que se aplican sobre estas imágenes. También se decidió definir clases para encapsular las llamadas a PVM (instanciación y comunicación entre procesos) para poder en el futuro cambiar esta librería por otra similar sin tener que modificar el resto del código. El encapsulamiento de los algoritmos en forma de clases permite agregar nuevos algoritmos en forma muy sencilla y siguiendo un esquema claro, además de permitir generar listas de algoritmos para aplicar secuencialmente a una imagen.

Clases para abstraer las imágenes y los pixels. La clase *Himage* encapsula una imagen RGB, la cual es una secuencia de pixels cada uno de los cuales posee tres componentes de color. Dichos pixels están encapsulados en la clase *Hpixel*. La clase *Himage* también encapsula un cursor (*Cursor*) que permite recorrer todos sus pixels en forma ordenada.

Clases para abstraer los algoritmos. Como ya se expresó, los algoritmos usados se encuentran encapsulados en clases. Para esto existe una clase virtual que es la superclase de todos los algoritmos (*HImageAlgo*), que define una interfase standard para todos ellos. El uso de cursores para recorrer la imagen facilita de manera significativa la implementación de los algoritmos, ya que permiten acceder a la información interna de la imagen de forma transparente e independiente del formato de la misma. También hay que resaltar que como es posible definir varios cursores al mismo tiempo sobre la misma imagen, la implementación de algoritmos mas complejos como el labeling o la detección de bordes también se simplifica, ya que por ejemplo se pueden definir cuatro cursores adicionales que recorran los vecinos cuatro-conectados del pixel que se está procesando.

Clases para abstraer los procesos master y clientes. El modelo de paralelismo elegido fue master-cliente: existe un proceso master que coordina el procesamiento y varios procesos clientes que realizan el procesamiento. Esto llevó a definir dos clases para encapsular estos procesos (*Hmaster* y *HClient*). La idea general de la aplicación es que se crea una instancia de un proceso *HMaster*, se le setean todos los parámetros necesarios (imagen a procesar, numero de procesos entre los cuales distribuir el procesamiento, valores de threshold, etc.) y se lo pone a procesar mediante el llamado a un método particular del objeto (process()). Este método instancia los procesos cliente correspondientes según los parámetros seteados, particiona la imagen entre estos procesos y luego coordina los resultados parciales de los mismos, para obtener el resultado final. Por otro lado, los procesos cliente, al ser ejecutados, crean una instancia de la clase *HClient* y lo ponen a procesar llamando al método process(). Este método espera recibir los datos, los procesa e intercambia resultados en forma sincrónica con el proceso master que lo instanció.

Clases para abstraer los procesos PVM. Básicamente existen dos tipos de procesos, con distintos requisitos de comunicación. Por un lado se encuentra el proceso master, que conoce a un conjunto de clientes con los cuales se comunica, y por otro lado, están los procesos clientes, que solo interactúan con el master que los instanció. Para modelar estos dos casos, se creó una clase llamada *HPvmProxi*, que representa a un proceso remoto. El master ve a cada cliente como una instancia de esta clase y a su vez los procesos cliente tienen una sola instancia de esta clase que representa al master. Esta clase tiene los métodos necesarios para comunicarse con el proceso remoto, y en particular guarda el identificador del mismo para poder enviarle y recibir mensajes de éste. Como PVM tiene funciones para enviar distintos tipos de datos (ints, longs, floats, etc.), esta clase tiene métodos *send* y *receive* sobrecargados con una implementación particular para cada tipo de datos. Por otro lado, se definió una clase llamada *HPvmProcessArray* que básicamente administra un número de proxis para facilitar la implementación del master.

7. Resultados obtenidos

En la evaluación de los resultados obtenidos debe tenerse en cuenta el hecho de que la aplicación fue desarrollada sobre una red de procesadores heterogéneos y con recursos limitados. Además, al estar trabajando sobre un entorno Linux, debe considerarse que éste realiza un scheduling de procesos particular que distribuye el tiempo de CPU entre la aplicación que se está ejecutando y los procesos de administración de recursos propios del sistema operativo. Esta distribución puede modificarse alterando las prioridades de los procesos, aunque de todos modos la aplicación no dispone del 100% del tiempo de procesamiento.

Para las pruebas se utilizaron dos procesadores Pentium (133 Mhz), con 32 y 48 Mb de RAM, y un procesador 80486 DX4 (100 Mhz) con 12 Mb. Para realizar las mediciones se armaron dos configuraciones de red distintas:

- El Pentium de 32 Mb con el 80486, utilizando placas de red NE-2000 compatibles de 10 Mbits/seg.

- Los dos Pentium con el mismo tipo de placas de red

Se realizaron sucesivas ejecuciones de la aplicación variando para cada una de ellas la cantidad de procesos clientes, la cantidad de hosts y la imagen a procesar. Además, cada caso fue ejecutado varias veces para disminuir los efectos del scheduling de memoria que realiza Linux. Al ir promediando las pruebas se notó que los tiempos de ejecución de la aplicación variaban sustancialmente entre corridas con cantidad de procesos clientes pares e impares, cuando la cantidad de hosts de la máquina virtual era par.

Analizando detenidamente cada prueba se observó que PVM distribuía en forma equitativa los procesos que ejecutaba entre los hosts de la máquina virtual. De esta manera, cuando la cantidad de clientes a ejecutar era impar, por ejemplo tres, y la cantidad de hosts era par, por ejemplo dos, PVM ejecutaba dos procesos cliente en un host y un cliente junto al master en el otro host.

Esta distribución de procesos provoca un aumento de los tiempos de ejecución: el host que procesa los dos clientes demora más que el otro pues el master lleva poco tiempo de ejecución. En el caso de tres clientes, debido a que la imagen a procesar se distribuye equitativamente entre los tres, el tiempo de procesamiento final será lo que demore en procesar dos tercios de la imagen el host con dos clientes. En cambio, si el total de clientes es cuatro, el tiempo de procesamiento final será lo que demore en procesar la mitad de la imagen el host con dos clientes y el master, siendo este último sensiblemente inferior al de la prueba con tres clientes.

Un ejemplo de la diferencia en los tiempos de ejecución entre corridas con cantidad de procesos clientes pares e impares se puede apreciar en la siguiente tabla (Nota: en todos los casos, los tiempos están expresados en segundos):

Procesos	1	2	3	4	5	6	7	8	9	10
Tiempo	6,18	5,85	5,48	5,1	5,77	5,55	5,95	5,76	6,43	6,28

Como consecuencia de este análisis, se decidió realizar las pruebas con cantidad de procesos clientes pares (además de la ejecución con un solo proceso cliente), para que los resultados obtenidos muestren claramente las diferencias en los tiempos de procesamiento al aumentar la cantidad de clientes (De todas maneras, si las pruebas se realizan con cantidad de procesos impares, los cambios en los tiempos de ejecución son porcentualmente similares).

Las primeras pruebas consistieron en ejecutar la aplicación sobre una máquina virtual de un solo host (en uno de los Pentium 133 con 48 Mb. de RAM). La imagen elegida para estas pruebas fue de una resolución de 264 x 384 pixels en 24 bits color. Los resultados obtenidos fueron los siguientes:

Procesos	1	2	4	6	8	10
Tiempo	4,36	3,695	3,47	3,71	3,865	4,33

La siguiente serie de mediciones se realizó luego de agregar a la configuración anterior un nuevo host (el otro Pentium), procesando sobre la misma imagen. Los resultados fueron:

Procesos	1	2	4	6	8	10
Tiempo	4,3	2,84	2,675	2,7	3,025	3,55

En la Figura 1 pueden apreciarse los tiempos de ejecución de las corridas realizadas con una y dos CPUs. La mejora obtenida considerando los mejores tiempos para cada caso fue de 1.29.

Para evaluar mejor el comportamiento de la aplicación con distintas cargas de trabajo, se realizaron las mismas mediciones sobre una imagen de mayor tamaño (522 x 804 pixels en 24 bits color). Los resultados obtenidos ejecutando la aplicación sobre una máquina virtual con un solo host fueron los siguientes:

Procesos	1	2	4	6	8	10
Tiempo	17,9	15,94	14,01	14,84	15,13	15,66

Al agregar otro host a la máquina virtual, se obtuvieron los siguientes resultados:

Procesos	1	2	4	6	8	10
Tiempo	16,95	8,206	10,77	10,84	11,05	11,6

En la Figura 2 pueden apreciarse los tiempos de ejecución de las corridas realizadas con una y dos CPUs. La mejora obtenida considerando los mejores tiempos en cada caso fue 1.7.

Para analizar el desempeño de la aplicación en ambientes heterogéneos, se configuró una red con el Pentium 133 Mhz y el 80486. Primero se ejecutó la aplicación sobre una máquina virtual con un único host (el 80486), obteniéndose los siguientes resultados:

Procesos	1	2	5	6	7
Tiempo	10,41	9,95	10,03	10,23	10,38

Agregando un host a la máquina virtual (el Pentium 133 Mhz) se lograron disminuir sustancialmente los tiempos de ejecución, siempre que al haber una cantidad impar de procesos clientes, la mayor cantidad posible (la mitad más uno) de ellos se ejecutara en el host más rápido. Los resultados fueron:

Procesos	1	2	5	6	7
Tiempo	7,12	6,65	4,7	5,82	5,76

En la Figura 3 pueden observarse los tiempos de ejecución de las corridas realizadas con una y dos CPUs heterogéneas. La mejora obtenida considerando los mejores tiempos para cada caso es de 2.11.

La siguiente prueba se efectuó sobre la misma configuración de red, pero con una imagen de 522 x 804 pixels en 24 bits, con lo que se espera obtener una notoria mejora de performance entre una ejecución con el 80486 como único host (caso I) y otra con el Pentium 133 como segundo host (caso II). Los mejores tiempos obtenidos fueron 35.9 segundos (para el caso I) y 15.3 segundos (para el caso II). La mejora fue del orden de 2.37.

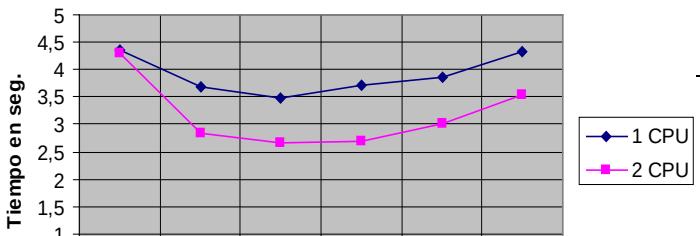

Figura 1.

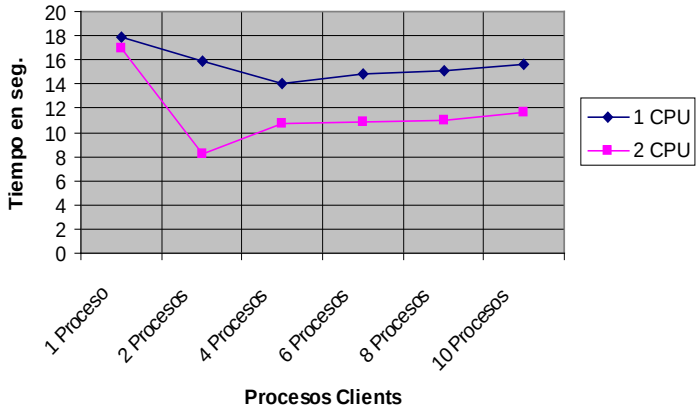

Figura 2.

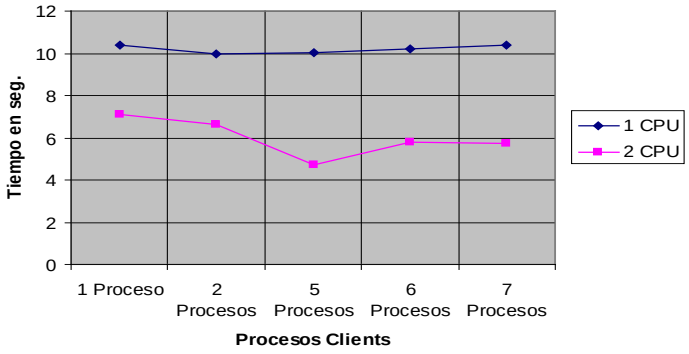

Figura 3.

8. Conclusiones y líneas de trabajo futuras

Se considera conveniente separar la evaluación entre el análisis de las herramientas utilizadas y los resultados obtenidos en sí mismos.Respecto de las herramientas, Linux resultó ser muy estable, lo que se traduce en una eficiente administración de los recursos de hardware; además provee facilidades para la configuración de redes TCP/IP (característica muy importante por la orientación de la aplicación), brinda una versión del popular compilador gcc de gnu para C y C++ (de prestaciones altamente satisfactorias), y con una buena implementación de PVM. En síntesis, la elección del entorno fue acertada para una aplicación de estas características (interconexión, concurrencia y manejo de grandes volúmenes de datos), además de permitir explotar la portabilidad.

Con relación a X-Windows, existen librerías free-ware que simplifican su uso; permite correr aplicaciones y ver los resultados en máquinas distintas, lo que facilita la realización de tests. Como características negativas, posee una API complicada y poco amigable, es difícil de configurar para poder utilizar altas aplicaciones gráficas, y la documentación está orientada a usuarios avanzados.

Por su parte, PVM hace totalmente transparente la comunicación entre los procesos distribuidos y la migración de configuraciones pequeñas a grandes (dado que la configuración de la máquina virtual es independiente de la aplicación que la utiliza). Una limitación la constituye el hecho de que no permite definir protocolos al estilo de Occam para enviar datos complejos como una estructura, lo cual hace necesario varias comunicaciones, cada una con un tipo distinto.

El lenguaje utilizado (C++), permite definir herencia entre clases y encapsulamiento de datos, lo que permitió realizar una implementación clara y fácilmente ampliable, y por su característica de bajo nivel permite una utilización óptima de los recursos disponibles, aunque requiere un excesivo cuidado en el manejo de la memoria y el debugger que provee la versión del compilador utilizado no es amigable, dificultando la detección de errores.

Los resultados obtenidos durante la etapa de mediciones de performance llevan a inferir que la implementación de soluciones utilizando paralelismo de grano grueso logra mejoras significativas sólo cuando los procesos involucrados son complejos (en el sentido de requerir un gran tiempo de procesamiento) y durante la ejecución no hay excesivas necesidades de comunicación. Esto se debe a que el costo de la comunicación es significativo con relación a la velocidad de procesamiento, pues se utilizó una red de área local.

Si bien los resultados obtenidos satisfacen altamente los objetivos planteados, pueden encararse líneas de trabajo alternativas con la intención de obtener mejoras en ciertos aspectos, como por ejemplo:

- Implementar un administrador de procesos que dinámicamente detecte los hosts más veloces de la máquina virtual (para evitar la descompensación en la carga de trabajo entre procesadores con distintas velocidades), y asignarles a éstos mayor carga de procesamiento, lo que se traduce en más imagen a procesar. De esta manera, se evitaría la distribución equitativa de procesos que realiza PVM, aprovechándose al máximo el tiempo de CPU disponible.

- Establecer una manera de asignar a los procesos de la aplicación la máxima prioridad de ejecución posible, de forma tal que Linux no dedique demasiado tiempo de CPU a sus procesos internos de scheduling.

- Utilizar otras configuraciones de red y sistemas de comunicación (por ejemplo fibra óptica) con el fin de bajar los tiempos de transferencia de datos entre procesos.

Además de estas mejoras, existen aspectos que no fueron contemplados, como el módulo de adquisición de la imagen desde la cinta transportadora, su digitalización y mejorado para su posterior procesamiento; o la etapa final de la aplicación encargada de evaluar los resultados obtenidos (en el procesamiento de una imagen determinada) y tomar decisiones en base a estos (por ejemplo, enviar el objeto en cuestión a donde el resultado de su clasificación indique).

9. Bibliografía

[Bax94] G. A. Baxes, "Digital Image Processing. Principles and Applications", John Wiley & Sons Inc., 1994.

[Cof92] M. Coffin, "Parallel programming- A new approach", Prentice Hall, Englewood Cliffs, 1992.

[CSA90] "OCCAM", Computer System Architects, 1990.

[Gon92] R. C. González, R. E. Woods, "Digital Image Processing", Addison-Wesley Publishing Comp., 1992.

[Har92] R. M. Haralick, L. G. Shapiro, "Computer and Robot Vision", Addison-Wesley Publishing Company, 1992.

[Hee91] D. W. Heermann, A. N. Burkitt, "Parallel Algorithms in Computational Science", Springer-Verlag, 1991.

[Hoa85] C. A. R. Hoare, "Communicating Sequential Processes", Pentice-Hall, 1985.

[Hus91] Z. Hussain, "Digital Image Processing", Ellis Horwood Limited, 1991.

[Hwa93] K. Hwang, "Advanced Computer Architecture. Parallelism, Scalability, Programmability", McGraw Hill, 1993.

[Jai89] A. Jain, "Fundamentals of Digital Image Processing", Prentice Hall Inc., 1989.

[Jai95] R. Jain, R. Kasturi, B. G. Schunck, "Machine Vision", McGraw-Hill International Editions, 1995.

[Law92] H. Lawson, "Parallel processing in industrial real time applications", Prentice Hall 1992.

[Lei92] F. T. Leighton, "Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes", Morgan Kaufmann Publishers, 1992.

[Lev90] S. Levi, A. Agrawala, "Real Time System Design", McGraw-Hill Inc, 1990.

[Mor94] H. S. Morse, "Practical Parallel Computing", AP Professional, 1994.

[PVM96] ¨Parallel Virtual Machine¨, World Wide Web.

[Rum91] J. Rumbaugh, M. Blaha, M. Premerlani, W. Lorensen, "Object-Oriented Modelling and Design", Prentice Hall, Englewoods Cliff, 1991.

[Sch92] R. Schalkoff, "Pattern Recognition. Stadistical, Structural and Neural Approches", 1992.

[Uma93] A. Umar, "Distributed Computing and Client-Server Systems", P T R Prentice Hall, 1993